# Notes on Project 3, Spring 2017:

## Tasks:

### Admin:

Github setup – Done by rmr5

Presentation slides

### documentation

Specification of the system

## System Specification

### General Approach

Dr. Fofanov suggested that he would try a seed based approach if he were solving the problem, unlike our idea of storing all genomic sequences for a given TaxID into a single prefix tree. He seemed genuinely intrigued about the approach of using a prefix tree at this time. He tried it 15 years ago and it did not turn out well, but with advances in compute technology, it might be worth another try. But one suggestion he did have was to put as many sequences as possible into the tree, where the terminator string for a sequence is it’s TaxID. This would have the advantage of allowing us to simply traverse to the end of the tree to find all TaxID’s that match a given query (if it hits), and also store as many possible subjects into the data structure at a time so that the time spend loading/unloading data from memory would be minimize.

As such, the current general approach would be to store all of the ***subject*** genomic sequences into a prefix tree, and take the queries directly from the fasta file and search them 1-by-1, recording results. It may also be possible (and wise) to add parallelism such that multiple queries could be searched against the subjects in the tree at the same time. Options include using a CPU with multi-threading, or a GPU with block launching.

### Details

## Code: (below are some ideas about how to potentially organize the code)

### Philosophy

1. Use Libraries where possible. For example, finding a Prefix Tree library that already exists and has full code functionality will save time, assuming it is implemented correctly for our needs.
2. Use camelCase naming convention for functions and “\_” naming convention for variables?
3. Regard the header files as an interface. This allows all code authors to know what to expect from functions that they are not implementing, but will be using.

### Organization of code

FASTA class:

- file reader

Prefix Trie class:

- Data Structure design and methods

- traversal/search (needs to start from all nodes in the tree)

- seed generation?? (if we choose something seed-based)

- fuzzy alignment logic

Reporting class:

- nicely report the binned data

Unit testing:

- all (or important) methods of the Tree class

Parallelism

- we should probably support threads at a minimum

- GPU threading would be nice on the K80 cluster, the problem lends itself well to parallelism